DA339A Inlämningsuppgift 4

Syfte

Inlämningsuppgift 4 examinerar inom alla lärandemål i kursplanen.

Inlämningsuppgift 4 utgör provkod 2006 Kursövergripande uppgift. Provkoden utgör 1 hp på kursen och har betygsskala UV vilket innebär att studenten kan erhålla betyg U, G eller VG på uppgiften. För att erhålla betyget VG så krävs att alla delar för G är uppfyllda samt även de delar som krävs för VG.

Den muntliga redovisningen är också en läraktivitet i kursen då studenten får återkoppling på inlämnad lösning. Vid redovisningen närvarar andra studenter som också ska redovisa och varje student lyssnar på andra students redovisning och återkoppling. Att se andras kod och andras lösningar är en viktig del i att lära sig programmera och detta är ett tillfälle för detta.

Gruppuppgift

Inlämningsuppgift 4 genomförs som en gruppuppgift i grupper om 2 personer. Det är tillåtet att göra gruppen ensam – en grupp med en person. Grupper anmäls via kursplatsen på Canvas genom att gå med i någon av grupperna Inlämningsuppgift 4 Grupp X. Om man avser göra uppgiften själv så måste man ändå gå med i en grupp - även om man blir den enda medlemmen i gruppen.

Det sker ingen anpassning av uppgiften beroende på om det är en eller två personer i gruppen.

Även om uppgiften genomförs i grupp så är examinationen individuell. Detta innebär att betyg inte sätts på gruppen som helhet (gruppbetyg är inte tillåtet vid Mau – se Studenters rättigheter och skyldigheter, kapitel 2.7) utan det görs en individuell bedömning av varje student vid den muntliga redovisningen. Detta innebär att en student i gruppen kan få ett godkänt betyg om hen kan svara på frågor och presentera lösningen bra medan den andra studenten kan behöva göra en komplettering om hen inte kan svara på frågor och förklara lösningen tillfredställande.

Gruppen måste tillsammans avgöra om man satsar på ett G eller VG då lösningen lämnas in gemensamt av gruppen.

Redovisning

Inlämningsuppgift 4 redovisas genom skriftlig inlämning i grupp på kursplatsen på Canvas samt en muntlig redovisning. Vid den muntliga redovisningen görs en individuell bedömning (se ovan) men varje grupp redovisar vid samma tillfälle/tid. Redovisningstid bokas för gruppen och samtliga gruppmedlemmar förväntas närvara denna tid.
Inlämning ska endast göras om gruppen har en komplett lösning som man anser uppfyller kraven för uppgiften.
Gruppen får inte lämna in en lösning som saknar delar och använda redovisningstiden för hjälp. Behöver man hjälp med inlämningsuppgiften så nyttjas handledningstider och/eller labbtider till hjälp för detta. De som försöker redovisa en lösning som saknar delar kommer att avbrytas och hänvisas till nästa hjälptillfälle för hjälp och nästa omtillfälle för inlämning/redovisning.

Skriftlig inlämning

I den skriftliga redovisningen ska följande lämnas in:

- En zip-fil med källkod som innehåller de filer och kataloger som ingår i projektet. De filer som lämnas in ska kunna kompileras och programmet exekveras. Struktur med kataloger för paket ska behållas i zip-filen men inga andra kataloger eller filer ska finnas i inlämningen.
- En pdf-fil med namn enligt mönstret *da339a_InUpp4_Grupp_x.pdf* (x ersätts med gruppens nummer) som innehåller
    - Förnamn och efternamn samt personnummer på gruppens medlemmar
    - Bilder med de diagram som efterfrågas i uppgifter nedan
    - De textsvar som efterfrågas i uppgifter nedan
    - Frågor till den andra gruppen som förbereds innan redovisning (se uppgfiter nedan)

Diagrammen får inte vara ritade för hand utan ska vara ritade med något verktyg exempelvis Visual Paradigm (men måste inte nödvändigtvis vara ritat med VP).

Deadline för inlämning till det ordinarie tillfället är **onsdag 13/1 23.55**. Inlämningar som lämnats in sent garanteras inte redovisning vid det ordinarie tillfället.

Om man inte lämnat in till det ordinarie tillfället eller blir underkänd vid den muntliga redovisningen för det ordinarie tillfället så ges det två omtillfällen med deadline för inlämning:

- Onsdag 3/2-2021 23.55 (tid för muntlig redovisning är ej satt ännu då det beror på vårens scheman)
- Augusti 2021, exakt datum ej bokat

Observera att detta är de enda examinationstillfällena som garanteras för Inlämningsuppgift 4.

Muntlig redovisning

För att göra den muntliga redovisningen så måste en skriftlig inlämning gjorts innan deadline. Endast kompletta lösningar får redovisas.

Vid redovisning så närvarar två grupper samtidigt. Grupperna kommer att behöva ställa frågor till varandra (se uppgifter längre ner) som en del av examinationen. Det är därför viktigt att gruppmedlemmar kommer i tid till sin bokade redovisningstid och stannar under hela det bokade redovisningspasset. Om någon kommer för sent eller lämnar för tidigt kan personen bli underkänd och behöver närvara vid ytterligare någon redovisningstid. Varje redovisning omfattar två grupper (4 studenter) och är planerade att ta cirka 60 minuter.

Har gruppen gjort en inlämning av den skriftliga delen så bokar man en tid för redovisning via kalendern för kursen i Canvas. Tider för muntliga redovisningar blir senast tillgängliga dagen efter deadline för den skriftliga inlämningen. Se separata instruktioner för hur du gör en bokning av muntligt redovisning.

För godkänt på uppgiften krävs att den muntliga redovisningen godkänns. Det är inte tillräckligt att endast ha lämnat in en skriftlig redovisning som löser uppgiften. Studenten måste kunna förklara sin lösning och hur denna fungerar och kunna svara på frågor om vad som händer i koden under vissa förutsättningar. En icke tillfredställande muntlig redovisning kan resultera i att kompletteringar av den skriftliga redovisningen kan behöva lämnas in.

Förberedelser

Inlämningsuppgift 4 omfattar kursinnehåll som gåtts igenom till och med föreläsning F24. I Inlämningsuppgift 4 så används flera olika byggstenar för problemlösning som tidigare används i laborationer och exempel.

Övergripande beskrivning

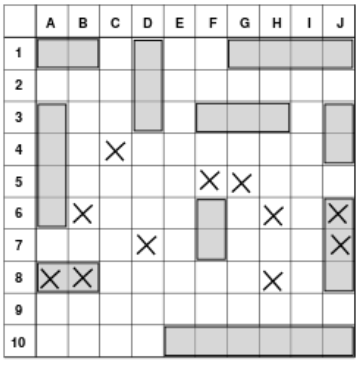
Sänka skepp är ett enkelt sällskapspel för två personer. Spelet består av en spelplan som kan vara olika stor men 10*10 rutor (total 100 positioner) är det vanligaste. På spelplanen placeras ett antal "skepp" ut. I vår version så har vi 5 skepp av olika storlek:

- u-båtar, en ruta stort
- torpedbåtar, 2 rutor stort
- jagare, 3 rutor stort
- kryssare, 4 rutor stort
- slagskepp, 5 rutor stort

Man kan som i exemplet nedan ha flera skepp av samma storlek eller att det inte finns några skepp alls av en viss typ.

De två spelarna börjar spelet med placera ut sina båtar på sina respektive spelplaner. Nu börjar den ena spelaren att gissa var den andra spelaren har lagt ut sina båtar genom att säga koordinaterna ex (D,3). Får man en träff så markerar man och meddelar motståndaren att han fått träff. När ett skepp blivit träffat i alla positioner det täcker så är skeppet sänkt. Den som första sänker alla motståndarens skepp vinner.

I bilden nedan så är de mörkare rutorna olika skepp. Skott representeras av ett kryss. Det finns ett sänkt skepp, en torpedbåt, på position (A,8) till (B8). En jagare har två träffar på positioner (J,6) och (J,7). Där finns flera skott som inte träffat något skepp.



För mer information om Sänka skepp se: http://hurspelarman.se/sanka-skepp/

Uppgifter för betyg G

För att få godkänt skall gruppen implementera en förenklad variant av spelet sänka skepp där endast en spelare spelar. Som poängsystem används hur många skott som behövs för att sänka alla skepp på spelplanen. Ju färre skott som behövdes desto bättre är resultatet. De tio bästa resultaten skall presenteras på en highscorelista där man skall kunna se poängen och namn på den som spelat.
Uppgiften ska implementeras som ett program i Java med ett GUI (Graphical User Interface) i Swing. Hur inmatning sker är upp till gruppen att avgöra inom kraven för funktion och implementation nedan. Highscore listan måste visas efter spelomgången är slut, men kan också visas hela tiden eller visas vid exempelvis ett knapptryck.
Spelplanen representeras av en array med objekt som representerar skepp finns och arrayen används för att veta var olika skepp är positionerade. För G ska två olika spelplaner finnas förberedda.
För godkänt ska det utöver den körbara källkoden finnas ett klassdiagram och ett sekvensdiagram enligt krav nedan. Uppgiften innehåller även tre textuppgifter som presenteras nedan.

Krav på implementation för betyg G

Följande krav ställs på din implementation av uppgiften:

Id för krav	Kravbeskrivning
IU4IG1	Källkoden ska vara väl formaterad med lämplig indentering som presenterats på kursen.
IU4IG2	Högst upp i alla .java-filer ska det finnas en kommentar som innehåller: • Förnamn och efternamn • Datorid • Vilket program man läser på den eller de personer som aktivt deltagit i att skriva koden i filen.
IU4IG3	Källkoden ska gå att kompilera och exekvera.
IU4IG4	Uppgiften ska vara löst på rimligt sätt utifrån de verktyg för problemlösning som presenterats på kursen till och med F24.
IU4IG5	Samtliga instansvariabler skall vara privata.
IU4IG6	Metoderna i controler- och entity-klasser skall dokumenteras kort ovanför metoddeklarationen gällande metodens syfte och namn, datatyp och kort beskrivning ges för varje parametrar samt datatyp och beskrivning för eventuellt returvärde (se L17 och Forumtest.java för exempel).
IU4IG7	Applikationen skall programmeras med en strikt MVC-struktur där view/boundary och model/entity inte skall känna till varandra (de ska inte importera varandras paket).
IU4IG8	Interface eller generalisering (eventuellt med abstrakt klass/metod) ska användas för att bygga en klasstruktur för olika typer av skepp i spelet (se kommentar under tips och ledning).
IU4IG9	Spelplanen ska representeras av en array med referenser till objekt för de olika skeppen. En plats utan skepp har värdet null.
IU4IG10	Två olika spelplaner ska vara förberedda i koden. Vilken spelplan som används kan avgöras genom “hårdkodning”, via val i GUIt eller som input vid uppstart av programmet.

Krav på funktionalitet för betyg G

Det program du skapar ska uppfylla följande krav på funktionalitet i programmet:

Id för krav	Kravbeskrivning
IU4FG1	Applikationen skall ha ett sammanhängande grafiskt gränssnitt (det vill säga ett huvud-fönster i form av ett JFrame-objekt och kan inte endast byggas upp med pop-up-fönster typ JOptionPane, JOptionPane kan däremot användas för enstaka saker som exempelvis felmeddelande)
IU4FG2	Spelet ska visa en spelplan med minst dimension 10*10 rutor.
IU4FG3	Det ska som minst finnas ett skepp av varje typ på spelplanen.
IU4FG4	Typer av skepp som ska finnas: • u-båt, en ruta stort • torpedbåt, 2 rutor stort • jagare, 3 rutor stort • kryssare, 4 rutor stort • slagskepp, 5 rutor stort
IU4FG5	Skepp får inte överlappa varandras positioner på spelplanen.
IU4FG6	Skepp ska kunna vara orienterade vågrätt eller lodrätt (inte diagonalt).
IU4FG7	Minst ett skepp måste finnas i varje orientering (vågrätt och lodrätt).
IU4FG8	Skepp måste ha en position där hela skeppet ryms på spelplanen.
IU4FG9	Varje skott som användaren gjort ska visas på spelplanen.
IU4FG10	Det ska inte gå att skjuta på en position två gånger.
IU4FG11	Varje skott ska indikera om det var en träff i ett skepp och vilken typ av skepp som träffats.
IU4FG12	När ett skepp blir sänkt så ska detta anges för användaren.
IU4FG13	Användaren ska kunna välja var denna vill skjuta nästa gång. Detta kan ske genom att klicka på spelplanen direkt eller genom att mata in positionen i textfält eller liknande lösningar.
IU4FG14	Spelet ska hålla reda på hur många skott som krävs för att sänka alla skepp i en spelomgång.
IU4FG15	Det ska finnas en highscorelista som innehåller namn och antal skott för de 10 bästa spelomgångarna.
IU4FG16	Higscorelistan ska vara sorterad i växande ordning med det resultat som krävde minst antal skott överst.
IU4FG17	Om antalet skott som krävdes för att sänka alla skepp kvalar in på highscorelistan ska användaren ombes ange ett namn vid spelets slut. Alternativt så anger spelaren ett namn innan spelomgången börjar och detta namn används om resultatet kvalar in på higscorelistan.
IU4FG18	Man ska kunna välja att spela en ny spelomgång via GUIt.

Krav på diagram för betyg G

Lösningen ska dokumenteras med ett klassdiagram och ett sekvensdiagram. Säkerställ att diagrammen som lämnas in stämmer överens med varandra och den kod som lämnas in.

Följande krav ställs på diagrammen:

Id för krav	Kravbeskrivning
IU4DG1	Klassdiagrammet ska följa den UML-notation som presenterats på kursen.
IU4DG2	Klassdiagrammet ska visa huvudklassen/klasserna för GUI-delen, dvs. Den eller de klasser som anropas av någon klass av typen Control. Alla control-klasser och alla entity-klasser skall visas i klassdiagrammet.
IU4DG3	Klassdiagrammet behöver inte ta hänsyn till att klasserna ligger i olika paket. Alla klasser visas i ett och samma klassdiagram utan paket i diagrammet.
IU4DG4	De klasser som ska visas i klassdiagrammet ska visas med alla attribut och namn och datatyp/klass för attributen. Operationer/metoder behöver inte visas i klassdiagrammet.
IU4DG5	För alla klasser i klassdiagrammet ska det anges om klassen är av typen boundary, control eller entity.
IU4DG6	Klassdiagrammet ska visa om en klass eller metod är abstract.
IU4DG7	Klassdiagrammet ska visa eventuellt interface som du skapat och som används.
IU4DG8	Associationer mellan klasser i klassdiagrammet ska visas med multiplicitet i bägge ändarna (där detta går att tillämpa) och lämpliga typer av associationer (exempelvis generalisering, aggregation eller komposition - överanvänd dock inte detta).
IU4DG9	Det ska finnas ett sekvensdiagram som visar vad som sker när användaren har valt att skjuta på en viss position och till och med att GUI:t uppdaterats med resultatet av det skottet.
IU4DG10	I sekvensdiagrammet ska alla parametrar i operationer/metoder visas med namn och datatyp/klass och returvärden ska visas med datatyp/klass.
IU4DG11	Sekvensdiagrammet ska visa alla anrop som sker mellan klasserna i klassdiagrammet för den aktivitet som sekvensdiagrammet gäller. Alla iterationer och selektioner i anropade operationer/metoder ska visas i sekvensdiagrammet.
IU4DG12	Sekvensdiagram och klassdiagram måste vara konsistenta – det vill säga de måste stämma överens och om ett anrop sker från ett objekt av klass A till ett objekt av klass B så måste klass A och B vara associerade i klassdiagrammet.
IU4DG13	Diagrammen ska vara ritade med något verktyg som Visual Paradigm eller liknande och får inte vara en handritad skiss på papper som sedan skannats in.

Textuppgifter för betyg G

I den skriftliga inlämningen (i pdf-filen) ska det finnas text som svarar på följande frågeställningar:

- **IU4TG1:Motivera valet av interface eller generalisering för att hantera olika typer av skepp.** Beskriv era motiv till varför gruppen valde det ena alternativet framför det andra och vilka fördelar ni såg med det valet. Beskriv även vilka nackdelar valet eventuellt medfört. Motiveringen ska baseras i fördelar och nackdelar kopplade till kodstrukturen/arkitekturen om implementeringen av uppgiften ("det kändes bättre" är inte ett tillräckligt bra motivering). Texten ska huvudsakligen bestå av redogörande löptext och får inte endast vara korta påståenden eller punktlistor. Texten ska också vara rimligt formaterad och formulerad gällande stavning och grammatik.
- **IU4TG2: En beskrivning av hur er lösning hade kunnat utökas för att kunna hantera två spelare som spelar mot varandra.** Beskrivningen ska innehålla vilka nya klasser ni tror hade behövts om lösningen istället skulle innebära att två spelare spelade motvarandra och försökte sänka skepp på den andras spelplan. Ni kan utgå från att spelarna sitter vid samma dator men turas om att titta på skärmen. Beskriv vilken påverkan en sådan lösning sannolikt skulle ha på kodstrukturen/arkitekturen av er lösning samt hur algoritmen för att spela ett spel hade behövts förändras. Texten ska huvudsakligen bestå av redogörande löptext och får inte endast vara korta påståenden eller punktlistor. Texten ska också vara rimligt formaterad och formulerad gällande stavning och grammatik.
- **IU4TG3: Förbered minst två frågor till den grupp som redovisar samtidigt.** Frågorna ska basera sig på era egna erfarenheter av uppgiften. Det kan vara frågor av typen "Vi hade problem med hur vi skulle lösa XXX. Hur löste ni detta?", "Vi hade svårt att välja hur vi skulle göra med XXX. Hur resonerade ni runt valet av XXX?" eller liknande.

Svar på frågeställningarna ovan redovisas också muntligt vid den muntliga redovisningen och tillfredställande svar krävs för godkänt. Frågorna ställs till den andra gruppen vid redovisning och varje grupp måste vara beredd på att svara på frågor av den här typen från den andra gruppen.

Uppgifter för betyg VG

För betyget VG så krävs att alla uppgifter för betyget G är tillfredställande lösta och redovisas tillfredställande samt att uppgifter för betyget VG nedan är lösta och redovisas tillfredställande.

Uppgifterna för VG är i stora delar samma som för G men gruppen ska :

- implementera en funktion som genererar en ny slumpmässig spelplan för varje spelomgång
- Ge en lösning för en algoritm för en dator-spelare.

Krav på funktionalitet för betyg VG

För betyget VG så ska den fasta arrayen för spelplanen som gäller för betyg G ersättas med funktionalitet som slumpmässigt genererar en fördelning av skepp i arrayen för spelplanen och kan användas för att spela flera på varandra följande spelomgångar med en ny spelplan varje gång. I övrigt gäller samma krav på fördelning av skepp på spelplanen och positionering av dessa som för G.

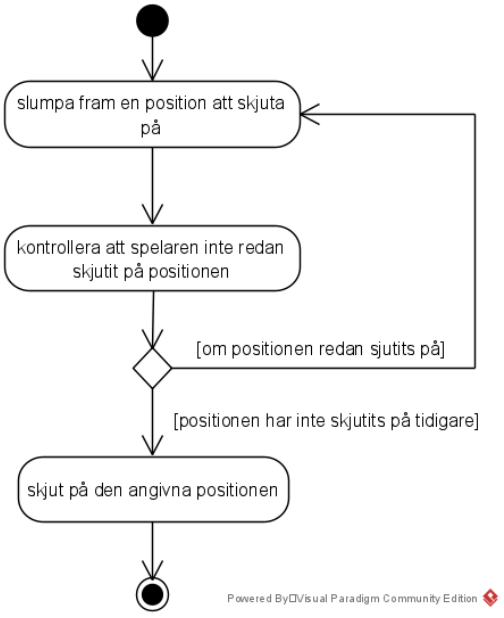
Lösningen ska uppfylla följande krav på funktionalitet för att kunna erhålla betyg VG:

Id för krav	Kravbeskrivning
IU4FVG1	Varje ny spelomgång ska automatiskt generera en ny spelplan enligt de krav som ställs på spelplanen för betyg G.

Diagram som krävs för betyg G ska stämma överens med den implementerade lösningen för VG om denna genomförs.

Textuppgift och aktivitetsdiagram för VG

För betyg VG krävs utöver den utökade funktionaliteten ovan att det i text och med ett aktivitetsdiagram beskrivs en algoritm för hur en dator-spelare (“AI”) hade kunnat skapas. Denna dator-spelare ska ha en bättre strategi än att helt slumpmässigt välja varje skott på spelplanen på en position som man inte skjutit på tidigare. Antag att den simplaste dator-spelaren har en algoritm som illustreras av följande aktivitetsdiagram:



Utöka aktivitetsdiagrammet till att beskriva en algoritm som minst tar hänsyn till punkterna i krav IU4DVG3 nedan. Aktivitetsdiagrammet ska visa den utökade algoritmens grundstruktur.

Följande krav ställs på diagrammet och algoritmen det ska visa:

Id för krav	**Kravbeskrivning**
IU4DVG1	Aktivitetsdiagrammet ska följa den UML-notation som presenterats på kursen.
IU4DVG2	Aktivitetsdiagrammet ska ha en detaljnivå motsvarande kod vad som sker i algoritmen. En jämförelse är en aktivitet. Kostare sekvenser av instruktioner kan sammanfattas som en aktivitet.
IU4DVG3	Algoritmen som beskrivs ska ta hänsyn till följande saker: • Resultat av föregående skott - träff/inte träff och om träff vad som träffades • Strategi för att välja en ny position om föregående skott inte var en träff beroende på om ännu tidigare skott var en träff eller inte. • Hur kan man göra ett bättre val än ren slump om man inte har ett identifierat skepp att försöka sänka?

IU4TVG1: Aktivitetsdiagrammet för algoritmen ska kompletteras med en textbeskrivning som förklarar varför algoritmen skulle ge ett bättre resultat än slumpmässig helt valda skott i de olika stegen. Texten ska även beskriva vilka förändringar som skulle behövas i lösningen för att man skulle kunna spela en människo-spelare och en datorspelare mot varandra (utökning av krav IU4TG2 ovan). Texten ska huvudsakligen bestå av redogörande löptext och får inte endast vara korta påståenden eller punktlistor. Texten ska också vara rimligt formaterad och formulerad gällande stavning och grammatik.

Tips och ledning

Försök inte göra allt med en gång! Gör bit för bit och räkna med att vissa saker kommer kanske att behöva ändras i din design när du kommit en bit på väg. Bygg upp programmet steg för steg för att ha bättre kontroll på vad som ger dig problem i nuläget. Hård koda inte storleken på arrayer och matriser utan se till att ha möjlighet att ändra. På så sätt kan man få ett överskådligt problem i början och skala upp det efterhand.

Glöm inte att spara ditt arbete ofta och ta även backup av det du gör på en plats så fort något fungerar bra.

Nyttja klassdiagram för att få översikt över dina klasser och börja implementera dem efter hand och undersök hur de behöver relatera till varandra. Vilka klasser är inte beroende av någon annan klass - börja med att implementera dessa och bygg ut en klass i taget. Skriv gärna små testprogram till din kod som skapar objekt av klasserna och ger dem värde för att testa din kod.

Om GUI

Du kan skissa ditt användargränssnitt på ett papper eller ett ritverktyg (eller Powerpoint, Word, Paint, etc.). Gränssnittet skall visa komponenter som hanterar nödvändiga input/output. Dessa i sin tur bör ge dig en bild av vilka instansvariabler (och även metoder) du behöver ha i olika klasser. Utifrån detta kan du designa Controller-klassen och GUI-klasserna. Tänk igenom de scenarier som du behöver genomföra i programmet (tänk på hur metoden för CRC-kort fungerar). Vilken information behöver finnas i GUI? Vilken information kan en controller-klass ta fram eller räkna ut givet vad den från GUI – och vad behöver då skickas till GUI från början? Komplettera din arkitektur efter hand med metoder som behövs för att genomföra de olika scenarierna. Tänk på att du nödvändigtvis inte behöver använda färger eller bilder på spelplanen utan att det går lika bra att använda bokstäver/tecken för att indikera status på olika sätt.

Gör en skiss av hur ni tänker er att GUIt ska fungera för att ta fram en strategi för logiken i programmets kommunikation. Hur GUIt utformas påverkar den övriga strukturen och tvärt om. Den skissen är dock inget som sedan slaviskt måste följas.

Tycker du att det svårt att implementera ett GUI vänta med det tills du har gjort en fungerade spelmotor. Under tiden är det OK att bara mata in och skriva ut resultatet till konsolen. Utskrifterna kan sen vara bra debugghjälp till när du bygger ditt GUI. Det blir också mycket enklare att designa ett GUI när men vet vad som behövs. Skulle ni märka att ni vill/behöver ändra strategi för strukturen när ni gör spelmotorn så är det bara att ändra GUIt för att matcha en ny strategi.
Försök inte få det 100% som ni tänkt från början, utan se till att det fungerar först och putsa på det sen. Passar inte GUI spelmotorn ändra på GUIt. Det samma gäller om er skiss inte fungerar tillsammans med Swing komponenterna.

Om man i tittar på program för Sänka skepp eller liknande spel som finns på nätet använder de flesta JButtons och det tycker vi också. JButtons sättas ihop i en JPanel och anväder Grid eller Flow layout. En fördel med att använda knappar är man har viss funktionalitet klar i knappen som att låsa, ändra färg och skriva text. Nackdelen är att man är ganska låst till ett speciellt utseende och beteende.

Flow layout
https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html
Grid layout
https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html
https://www.geeksforgeeks.org/java-awt-gridlayout-class/

Eftersom vi har en spelplan som är minst 10*10 betyder det att vi skall skapa åtminstone 100 knappar. För att detta inte skall var en oöverstiglig uppgift så kan vi använda for-loopar och populera en matris av knappar. Sök på "create 100 jbutton java" och "java create jbuttons dynamically" för att få lite inspiration. Kom ihåg att göra lösningen konfigurerbar, så börja inte med att generera 100 knappar utan börja med en mindre storlek som 4*4 så blir det lättare att testa och arbeta på.

När matrisen är klar kan ni koppla ihop den med logiken.

Om logik och kodstruktur

För att hantera olika typer av skepp så kan man använda ett interface för de metoder som gäller alla skepp men som kan fungera lite olika. Varje typ av skepp representeras sedan av en egen klass som implementerar interfacet. Ett annat tillvägagångssätt är att använda en generell klass/superklass som innehåller det som är gemensamt för alla skepp. Detta kan eventuellt var en abstrakt klass med abstrakta metoder. Varje typ av skepp representeras sedan i form av subklasser.

För skepp så är det inte nödvändigt att veta exakt vilken position som träffats. Vid en träff i ett skepp så kan man räkna ner eller upp antalet träffar i förhållande till skeppets storlek. När man nått 0 eller maxantalet så är skeppet sänkt.

I den array som representerar spelplanen så kan du låta denna vara en array av skeppsobjekt. På varje position i arrayen som ett skepp ligger så har du en referens till det skeppsobjektet. Om där inte ligger något skepp så låter du positionen i arrayen ha värdet null. Vid en skott på en positionen kan du då enkelt avgöra om något skepp träffats och om så vilket skepp. Vid träff i ett skepp så kan referensen till skeppet i arrayen för den aktuella positionen tas bort så man inte kan träffa hela skeppet genom att skjuta på samma position igen.

En lösning för att hindra att man skjuter på samma position flera gånger är att om man använder knappar för spelplanen avaktivera (sätta enable till false) knappar som redan använts för skott på en position.

Beroende på hur man vill lösa att man inte ska kunna skjuta på samma position två gånger så skulle arrayen också kunna användas för att kontrollera om man tidigare skjutit på en viss position. Arrayen för spelplanen skulle då kunna bestå av mer generella objekt eller interface som kan svara på om de är en tom position om det blev en träff i ett skepp eller om det är en position som redan skjutits på. Det kan då räcka med ett objekt för tom position och ett objekt för redan skjutit på. Alla tomma positioner refererar till samma objekt och alla positioner man redan skjutit på referera till samma objekt. Vi en träff i ett skepp så kan man justera skeppets statur och sedan ändra referensen i arrayen till att peka på objektet för redan skjutit på.